GEN-R: Genetic Algorithm Based Model for Rubik's Cube Solution Generator

Ashutosh Tyagi
Department of Information Technology
JSS Academy of Technical Education
Noida, India
E-mail: ashutoshtyagi@live.in

Poonam Tyagi
B.Ed. Department
Apex Institute of Management Studies and Research
Meerut, India
E-mail: poonamtyagi134@gmail.com

***Abstract*-Evolutionary algorithm's implementation has solved problems in various disciplines including combinatorial optimization. Genetic algorithms especially have been able to provide more robust support for such problems. This work describes a model for generating solutions for a Rubik's cube based on genetic algorithm. Rubik's cube is a problem pertaining to discrete optimization. The proposed model named Gen-R takes scrambled state of a Rubik's cube as input. After implementation of a refined genetic algorithm and performing suitable optimization, the steps required to obtain the solved state are provided to the user. The model is flexible enough to adjust parameters of the algorithm as per the requirement. The experimental results obtained using Gen-R clearly establish relationships between parameters used in this model. Also the implementation of the model developed shows that it can effectively solve scrambled cubes and provide correct sequence of maneuvers.**

***Keywords*- Genetic Algorithm; chromosomes; gene; Rubik's cube solution factor; solved Rubik's cube factor.**

I. I. INTRODUCTION

Rubik's cube is an interesting puzzle in computer science with respect to group theory. The number of permutations that can arise from a scrambled Rubik's cube is enormous. Therefore attempts have been made to find the correct sequence of steps to solve the cube with the help of computers. Although few algorithms exist to solve Rubik's cube, yet development of computer models with the help of genetic algorithms can provide a more robust approach for solving these cubes. Genetic algorithms have evolved to serve as important constituents of applications dealing with metaheuristics. Genetic algorithms are part of evolutionary algorithms, so they tend to provide more optimized solutions to problems with the use of techniques inspired by natural evolution.

The model developed named Gen-R is based on genetic algorithm. The model also utilizes the algorithm presented by Denny Dedmore [1]. Denny Dedmore's algorithm was a milestone in providing solution for Rubik's cube, yet the possibility of computing the next move on the basis of available large number of available moves was a challenging task. The use of concept of genetic algorithm along with this approach provides a more lucid model. The model is tuned with certain parameters used in genetic algorithm so as to provide real time solutions to scrambled Rubik's cube.

This paper is outlined as follows. Section II consists of the related work done previously. Section III provides the detailed description of the model. Results are reported in Section IV. The work is concluded along with mention of future work in Section V.

II. RELATED WORK

A. Genetic algorithms

Genetic algorithms are inspired by Darwin's theory about evolution. Genetic algorithms (GA) are search algorithms based on the principles of natural selection and genetics. The most important idea that stands beyond the initial creation of genetic algorithms is the aim of developing a system as robust and as adaptable to the environment as the natural systems.[3] GA-based techniques are appropriate for dealing with situations having many candidate solutions rather than a single solution and

employ stochastic operators to guide the search process.[4] The GA is well suited to and has been extensively applied to solve complex design optimization problems because it can handle both discrete and continuous variables, nonlinear objective and constrain functions without requiring gradient information[5]. The genetic algorithm differs from other search methods in that it searches among a population of points, and works with a coding of parameter set, rather than the parameter values themselves. It also uses objective function information without any gradient information. The transition scheme of the genetic algorithm is probabilistic, whereas traditional methods use gradient information. Because of these features of genetic algorithm, they are used as general purpose optimization algorithm. They also provide means to search irregular space and hence are applied to a variety of function optimization, parameter estimation and machine learning applications. Genetic algorithms perform search in complex, large and multimodal landscapes, and provide near-optimal solutions for objective or fitness function of an optimization problem [6].

Outline of the Basic Genetic Algorithm

A. Generate random population of n chromosomes (suitable solutions for the problem).The encoding of the chromosomes can be binary encoding, permutation encoding, value encoding, tree encoding, etc.

B. Evaluate the fitness f(x) of each chromosome x in the population.

C. Create a new population by repeating following steps until the new population is complete.

i. Selection - Select two parent chromosomes from a population according to their fitness to survive. This selection can be-

> Roulette Selection: Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have.

> Rank Selection: The previous selection will have problems when the fitness differs very much. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population). After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

>. Steady State Selection: In every generation a few (good - with high fitness) chromosomes are selected for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

ii. Crossover- With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

iii. Mutation- With a mutation probability, mutate new offspring at each locus (position in chromosome).

iv. Accepting- Place new offspring in a new population.

D. Test if the end condition is satisfied, stop, and return the best solution in current population otherwise goto step B.

The algorithm flow is shown in figure 1.

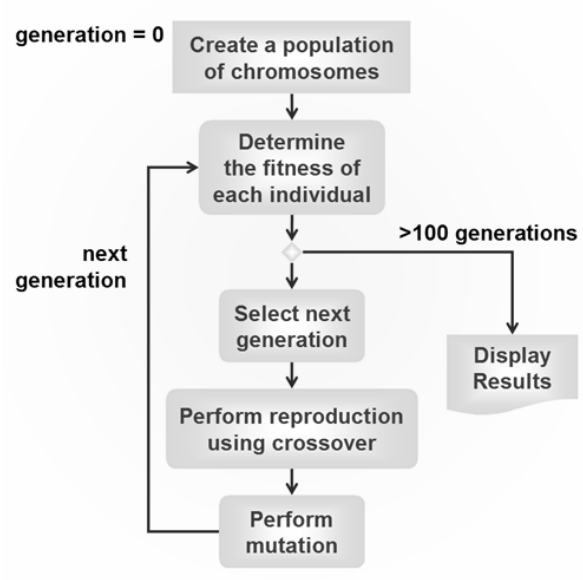

Figure1. Basic Genetic Algorithm

B. Rubik's cube:

An l*m*n Rubik's Cube is composed of lmn cubelets, each of which has some position (x, y, z), where x ε {0, 1…..l-1}, y ε {0, 1…..m-1}, z ε {0, 1….nl-1}.Each cubelet in a Rubik's Cube has a color on each visible face. There are six colors in total. We say that a Rubik's Cube is solved when each face of the cube is the same color, unique for each face. [7] Each of these cubelets includes a concealed inward extension that interlocks with the other cubes, while permitting them to move to different locations. However, the center cube of each of the six faces is merely a single square façade; all six are affixed to the core mechanism. These provide structure for the other pieces to fit into and rotate around. So there are twenty-one pieces: a single core piece consisting of three intersecting axes holding the six center squares in place but letting them rotate, and twenty smaller plastic pieces which fit into it to form the assembled puzzle [8].

Rubik's Cube Solving Methodologies:

The chief methodologies for solving the Rubik's Cube are given by Denny Dedmore, Thistlethwaite, Kociemba etc. Their algorithms are based on group theory concepts and employ various variations of traversal methods. Thistlethwaite's Algorithm is based on dividing the problem into sub problems and solving them. By using already calculated lookup-tables, sequences are put together that move a Cube from one group into another until it is solved [13].

Kociemba's algorithm divides the problem into subgroups as in Thistlethwaite algorithm, but reduces the number of needed subgroups. This method uses an advanced implementation of IDA, generating small maps, calculating and removing symmetries from the search tree and tends to solve the Cube close to the shortest number of moves possible. Kociemba made his approach available in form of a program called Cube Explorer. Denny Dedmore's algorithm has been widely used because of its simplicity in implementation. The whole process of finding a solution from the available scrambled configuration is divided into certain stages which are solved individually to obtain the required solution.[1]

III. PROPOSED MODEL

A. *System Interface*

The system is designed using java. The interface of this system is used to enter the current scrambled state of the Rubik's cube. The current scrambled state is input as a binary string. The binary string contains the information regarding the position of all cubelets on the cube. It is done by encoding each color as binary number. All positions on the cube are encoded with respect to center cubelets. Only certain cubelet values are required to be provided as other values can be determined from the obtained data. Along with this other parameters which are required in the working of genetic algorithm such as mutation rate and crossover rate can also be determined. If the crossover rate is negligible then the generated offspring will be similar to the parents. The crossover rate can be set from a range of 0 %-100 %. Also the mutation rate can be set within the range of 0%-50%. Mutation is helpful in preventing the clustering of all solutions in to small local optimum set of solved problems. Population size and number of generations and required fitness percentage can also be set here. Figure 2 demonstrates an interface of the following system.

Figure2. Interface of the system

B. *Overview of the Algorithm:*

1. Determine the initial state of the Rubik's cube along with crossover rate, mutation rate and population size.
2. Compute Rubik's cube solution factor and set variable counter to 7.
3. Generate random population of chromosomes using binary encoding.
4. Select chromosomes satisfying the prescribed fitness rate using roulette selection.

5. Perform two point crossover of the selected chromosomes to generate new chromosomes.
6. Perform mutation in the generated chromosomes as per the mutation rate.
7. Go to step 3, until average of resultant Rubik's cube' solution factor has improved by more than 50%.
8. Set counter value to counter-1. If counter value equals zero, go to step 9. Otherwise go to step 3.
9. Decode the chromosome to obtain the required maneuvers to obtained solved state, display the result and exit.

We describe the algorithm in more details below.

C. Rubik's Cube Solution Factor:

Rubik's cube solution factor is a measure of the magnitude of Rubik's cube solved. It determines the ratio of correct cubelets in their position with respect to a solved Rubik's cube cubelets.

$$\text{Rubik's Cube Solution Factor} = \frac{\sum_{i-1}^{n} \text{location}[i]*\text{facecolor}[i]}{\text{solved rubik's cube factor}}$$

Here, i refers to cubelet of Rubik's cube, location refers to any specific position of cubelet on the cube and facecolor refers to the color of the cubelet. The numerator value is termed as current Rubik's value. Solved Rubik's cube factor is the value obtained when Rubik's cube solution factor becomes 1 i.e. when the cube is completely solved.

D. Structure of a chromosome:

Each chromosome in the given population consists of twenty genes. Each gene consists of all possible moves that can be made on a Rubik's cube in one turn. The first six bits refer to horizontal movement. Value 01 means movement form left to right and value 10 means movement form right to left and 00 or 11 means no movement. The next three bits shows vertical movement. Value 10 means top-down movement value 01 means bottom-up movement. 00 or 11 mean no movement. The last bit shows complete movement of cube in anticlockwise (0) or clockwise (1) direction.

E. Selection of chromosomes:

Chromosomes are selected on the basis of their fitness factor using Roulette selection. Fitness factor is equal to the degree of agreement to Rubik's cube solution factor. Better the fitness factor more be the chance of selection of chromosomes to survive in the population.

F. Crossover:

Two point crossover of the chromosomes is performed by selecting two crossover points. Crossover is initiated from half of the length of the chromosome to the end of the chromosome. If the crossover rate is more than 50% then after reaching the end of chromosome, the crossover starts from the beginning of the chromosome to the desired location on the chromosome.

G. Mutation:

Single bit mutation is used in the given model. Random bits are selected across the chromosome length according to mutation rate and these bits are inverted to generate new mutated chromosomes.

New generations are generated till the average of the Rubik's cube solution factor of the chromosomes has not improved by less than 50%. According to Denny Dredmore's [1] algorithm a Rubik's cube can be solved within a range of steps consisting of varied maneuvers depending on the present state of scrambled cube. Therefore each time the required generation achieves its target the value of the variable counter is reduced by 1. As the counter value is reduced to 0, the final sequence of steps which leads to solving the initial scrambled Rubik's cube to the final solved state is displayed on the interface as discussed in system interface section.

IV. RESULT

The graph plotted between current Rubik's value and solved Rubik's cube factor in percentage as compared to itself is shown in figure 3.

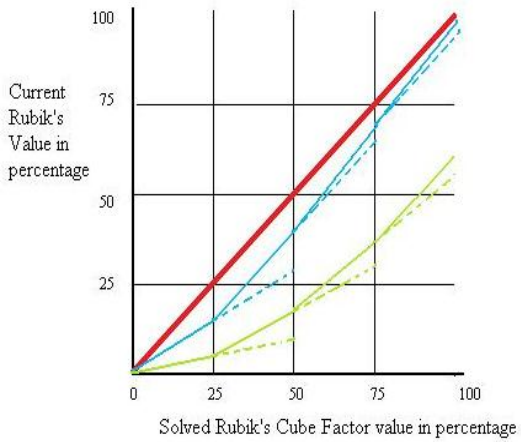

Figure3. Relationship between current rubik's values to solved rubik's cube factor value

The red line depicts optimum case when the cube is already solved. The blue and green lines on the graph depict two scenarios case I and case II respectively of differently scrambled Rubik's cube. As the case I has higher Rubik's cube solution factor as compared to case II so in this case the chromosomes have higher fitness rate. Also the graph shows that the convergence of case I to optimum case is more as compared to case II. Thus it is clear that if the initial generated population has high fitness rate then the accuracy of optimal solution improves.

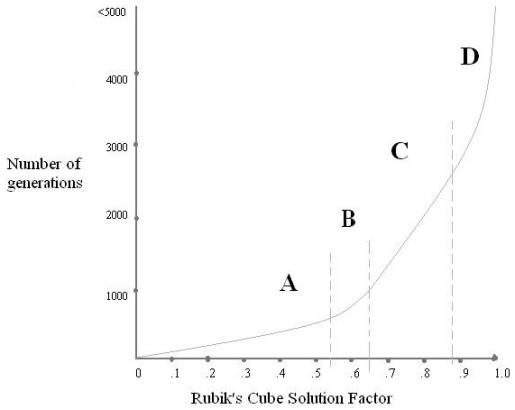

Figure4. Relationship between number of generations to rubik's cube solution factor

Figure 4 represents the graph plotted between Rubik's cube solution factor (shown on axis x) and number of generations required to achieve corresponding Rubik's cube solution factor. (shown on axis y). The graph data is plotted with crossover rate of 50% and mutation rate of 0.5% for all cases. The graph can be divided into four segments. Segment A denotes rapid improvement in value of Rubik's cube solution factor with generations. Segment B shows initiation in slow down of increase in factor's value. Segment C is almost uniform and section D shows that a large number of generations are required for obtaining very high values of Rubik's cube solution factor.

V. CONCLUSION

Gen-R is an effective model in determining the correct sequence of steps required for solving Rubik's cube with various degrees of scrambles. The results obtained using this model shows a clear relationship between tuning parameters used in this model. The results also shows that selection of chromosomes with higher values of the Rubik's cube solution factor in the initial stages leads to better solutions. The Gen-R can incorporate various rates of crossover and mutations for optimization. In future, the model can be improvised by providing user friendly entry of scrambled state of the cube. This model can also be extended to include other kinds of Rubik's cube with various configurations and number of cubelets.

REFERENCES

[1] Essay on the solution of Rubik's Cube, http://www.oppapers.com/essays/Rubic-s-Cube-Solution/517545, 2010.

[2] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publications, 1989.

[3] Boncovic et all, "*A genetic algorithm based solution for Intrusion detection*", Journal of information assurance and security, 2009, pp: 192-199.

[4] G. Weiss, "*Mining with rarity: A unifying framework*",SIGKDD Explorations 6(1), 2004, pp:7-19.

[5] A. Varsek, T. Urbancic, B. Filipic, "*Genetic Algorithm in Controller Design and Tuning"*, IEEE Trans. System, Man, and Cybernetics, Vol. 23, No. 5, 2003, pp: 1330-1339.

[6] Ujjwal Maulik, Sanghamitra Bandyopadhyay, Genetic algorithm-based clustering technique, Pattern Recognition Society. Published by Elsevier Science Ltd., 2000 pp: 1455-1464.

[7] Erik D. Demaine et al, *"Algorithms for Solving Rubik's Cubes"*, 19th Annual European Symposium on Algorithms, 2011.
[8] Black, M. Razid; Taylor, Herbert, *Unscrambling the Cube*, published by Zephyr Engineering Design, 1980.
[9] Nail El-Sourani et. al., *"An Evolutionary Approach for Solving the Rubik's Cube Incorporating Exact Methods*", Genetic and Evolutionary Computation Conference (GECCO), 2010.